Modcomp Version of Tutorial Input

K. I. Moyd
Communications Systems Research Section

The version of Tutorial Input implemented on the Modcomp used for antenna control at DSS 13 is described. Emphasis is on the use of the Tutorial Input; program operation is described to the extent that it makes the use more understandable. Flow charts are provided.

I. Introduction

Tutorial Input, a standardized man/computer interface developed by A. I. Zygielbaum (Ref. 1) and modified by K. I. Moyd (Ref. 2), has been implemented on the Modcomp II computer used for antenna control at DSS 13. It is implemented as a set of FORTRAN subroutines that communicate with the user program by means of tables in common. Several features have been added to the version used in the DSS 13 automation project (Refs. 3 and 4).

II. Operation

Tutorial Input is divided into commands and parameters associated with the commands. Once the operator starts entering a command, he must enter values for all parameters associated with that command in a specified order. By doing a carriage return before all parameters are entered, the operator can request prompting of the next parameter to be entered.

The commands and parameters are defined in the program by means of tables set up by the user. A command definition includes the name of the command, the number of associated parameters and an index telling where the parameter definitions for the command start. A parameter definition includes the name of the parameter, a code indicating the type of the parameter, and an index telling where the parameter value is to be stored. Because of the difference in the number of words required to store floating point and integer numbers, there are actually two separate parameter buffers. However, for purposes of knowing how to use the program, it is acceptable to consider that there is a single parameter buffer. There is also a flag table that indicates to the user program whether a particular command has been entered. The flag table allows commands having no parameters to be used for program control. The command and parameter definition tables, the flag table, and the parameter buffer can be accessed by both the user program and the Tutorial Input routine. To prevent premature transfer of flags and parameter values, an internal flag table and parameter buffer are established within the Tutorial Input routine. The internal values are transferred to the user only upon normal termination of the Tutorial Input routine. This allows the operator to verify or cancel entries before transferring them to the user. Several "internal" commands have been implemented to allow for verification and cancellation.

The Tutorial Input routine is activated by the operator entering a line of input on the Terminet. Upon activation, the values in the user parameter buffer are transferred to the internal parameter buffer and the internal flag table is zeroed.

Input lines are interpreted from left to right by field. A field consists of the characters from the first nonblank character following the previous delimiter to the trailing delimiter, which may be a slash, blank, comma, or carriage return.

There are two kinds of fields, i.e., command and parameter. A command field is alphanumeric. The first four characters are used for comparison with the user-defined and internal command names. These four characters must match exactly, including blanks (trailing blanks will be added automatically if fewer than four characters are entered). If the user-defined command name has fewer than four nonblank characters, only that many characters may be entered by the operator. However, if the user-defined command name has four characters, the operator may enter as many characters as he wishes, as long as the first four agree. This difference is caused by the fact that blanks serve as delimiters.

A parameter field is interpreted according to the type specified by the user-defined code. There are four types: integer, double precision floating point, two-character alphanumeric, and six-character alphanumeric. An integer field, i.e., a field to be interpreted as an integer, may contain a decimal integer with sign (plus is optional) of magnitude less than or equal to 32767 or a four-digit hexadecimal number preceded by a #. A floating point field may contain an integer as described above, in which case the decimal point is assumed to be at the end, or it may contain a decimal number with an included decimal point. The field may not exceed eight characters. Any violation of these rules will cause an error message to be printed.

In the case of the alphanumeric fields, the number of allowed characters is not limited; however, only the first two or six characters will be transferred to the user. The same limitations may occur as for command fields if the user compares the parameter with a value containing fewer than the allowed number of characters.

The routine interprets all fields as command fields except when it is in the process of accepting the parameters for the previously entered command. When interpreting a command field, its first four characters are compared with the internal and then user-defined command names. If no match is found, an error message is typed, the remainder of the line is ignored, and the routine waits for a new line. If it matches with a user command name, the routine will then look for the parameters, if any. The operator may have the routine prompt him with the name of the first parameter to be entered by doing a carriage return after the command name. If he ends the line before all parameters are in, he will be prompted with the name of the next one to be entered. If the operator enters an * in place of a parameter value, the value in the internal buffer is left unchanged. Parameters will be converted to the format specified by the user code and stored in the internal buffer. Once all the parameters for a command have been found, an internal flag is set indicating that the command has been entered. The routine then checks whether there is additional input on the line. If so, it continues processing the line interpreting the next field as a command. If there is no additional input on the line, the user flags corresponding to the set internal flags are set, and the parameter values for the entered commands are transferred from the internal parameter buffer to the user parameter buffer. The routine is then deactivated until a new line is entered.

The internal commands are acted upon as soon as they are found. The EXIT command causes a normal termination to occur whether or not there is subsequent input on the line. All flags and parameters for commands preceding the EXIT are transferred to the user. The NOEX command prevents normal termination until another line has been entered. Anything on the same line following the NOEX is ignored. 'ENTER INPUT' will be typed to remind the operator. The LIST command causes all command names to be typed out with the associated parameter names and parameter values from the internal parameter buffer in a format corresponding to the user-specified type. If no input follows the LIST command, a normal termination will be made. Both the TYPE command and the DELE command require a user command name as a parameter. If the TYPE or DELE is not followed by any input, a normal termination will be made. The DELE command causes the internal flag for the command following the DELE to be set to a specific value different from the set or reset condition. Upon normal termination the corresponding user flag is reset. This allows a previously entered command to be deleted if it has not already been acted upon. It should be noted, however, that the values in the user parameter buffer will not be changed. If no additional input follows the command name, a normal termination will be made. The TYPE command causes the parameter values associated with the following command to be typed out from the internal buffer. Even if no additional input follows the command name, the routine will not be terminated. In this case 'ENTER INPUT' will be typed out. This feature allows the operator to verify the correctness of parameter entry before the values are transferred to the user. He may correct the values by repeating the command. The * may be used for any parameters that had been entered correctly. If the parameters were entered correctly to begin with, the EXIT command may be used to cause normal termination.

III. Additional Features

There are some additional editing features. The Terminet backspace (BS) and delete (DEL) keys may be used to delete a character or a line, respectively. If a $<$ is entered in place of any command name or parameter, the routine will be terminated immediately upon its interpretation without transferring any values to the user.

Error messages indicate the position and kind of error. An ! is typed out directly below the first character of the command or parameter in error. (Output caused by a preceding LIST or TYPE command may intervene.) There are two different error messages – UNRECOGNIZED COMMAND and ILLEGAL PARAMETER. In the first case, the routine expects a new command name to be entered; in the second case, the operator is prompted with the name of the parameter in error. In both cases all input preceding the error has been accepted; all input following the error will be ignored.

As an example, the commands and parameters being used in the DSS 13 antenna control program are presented in Appendix A. Because of the way Tutorial Input is set up, new commands may be implemented easily. It is expected that CONSCAN and three-day fit modes will be added in the near future.

Tutorial Input, as it is implemented here, could be used on any Modcomp with FORTRAN programming. Implementation details and flow charts are presented in Appendix B. Listings and additional material can be obtained from the author.

References

1. Zygielbaum, A. I., " 'Tutorial Input' – Standardizing the Computer/Human Interface", in *The Deep Space Network Progress Report* 42-23, pp. 78-86, Jet Propulsion Laboratory, Pasadena, California, October 15, 1974.
2. Moyd, K., "Fortran Implementation of Tutorial Input", in *The Deep Space Network Progress Report* 42-24, pp. 88-99, Jet Propulsion Laboratory, Pasadena, California, December 15, 1974.
3. Moyd, K. I. "Automatic Control of DSS-13", in *The Deep Space Network Progress Report* 42-29, pp. 107-114, Jet Propulsion Laboratory, Pasadena, California, October 15, 1975.
4. Moyd, K. I., "Remote Automatic Control of DSS-13", *The Deep Space Network Progress Report* 42-30, pp. 174-183, Jet Propulsion Laboratory, Pasadena, California, December 15, 1975.

Appendix A
Tutorial Input Commands for the DSS 13 Antenna Control Program

The first name is the command name; the indented names are the parameter names. b = blank (may be omitted but not replaced by nondelimiter).

Command	Parameter	Description
SIDb		Track a source at the sidereal rate. When acted on, RA and DEC will be typed in decimal degrees.
	ID	Up to 6 alphanumeric character identification.
	RA	Right ascension in form HHMMSST where HH = hours, MM = minutes, SS = seconds, T = tenths of seconds. Interpreted as double precision floating point.
	DEC	Declination in form ±DDMMSS where DD = degrees, MM = minutes, SS = seconds and the + is optional. Interpreted as double precision floating point.
AZEL		Move the antenna to a specified AZ and EL.
	AZ	Azimuth in decimal degrees 0.000 to 359.999 (decimal point to be included).
	EL	Elevation in decimal degrees. 0.000 to 90.000 (decimal point to be included).

	REG	Lb = left wrap-up region.
		Rb = right wrap-up region.
		other = center region.

OFFb		Change variable offsets (default values are zeroes).
	AZOF	Azimuth offset in decimal degrees.
	ELOF	Elevation offset in decimal degrees. Note: In SID mode, the azimuth offset will be divided by COS(EL) to get the specified number of degrees on the sky. In AZEL mode, the azimuth offset will be applied directly.
CLRO		Clear both variable offsets.
STOP		Safely decelerate antenna and apply brakes.

Appendix B
Implementation and Flow Charts

The implementation of Tutorial Input on the Modcomp was made easier by the fact that several string handling and conversion services are provided by the operating system (as REX services). Although some of these services have FORTRAN-callable versions, the assembly language versions had to be used to allow enough significant figures for the numerical parameters. FORTRAN-callable subroutines were therefore written to use these services. Several of the services are set up such that the output from one is in the form of the expected input of another. Details of these services can be found in the Modcomp computer manuals.

The main Tutorial Input routine is called TUTOR (Fig. B-1). It may be set up as a separate task or as a subroutine called by a user task. The remaining subroutines used to implement Tutorial Input are called only by the Tutorial Input routines, not by the user.

TUTIN (LFIRST) (Fig. B-2) provides one line of Terminet input to the calling program. If LFIRST is false, the input is accepted from the Terminet. If LFIRST is true, it is assumed that the input has already been accepted. This procedure is used so that the first line of input can be accepted by the user program before TUTOR is called.

TYPOUT (I, param buffers) (Fig. B-3) types out the internal values of the parameters for the I^{th} command in the formats specified by their type codes except that double precision floating point numbers are typed out both in double precision form with exponent, and in fixed point with three decimal places. The latter is easier to read, but may not contain enough significant figures.

COLLECT (IERR, LNUM) (Fig. B-4) uses the REX service COLLECT to pick out the next field in a character string. Delimiters are trailing blanks, slashes, commas, equal signs, and trailing plus or minus signs. If the field contains only characters allowed in numbers (decimal or hexidecimal if preceded by a #), LNUM is set to TRUE; otherwise it is FALSE. If a numerical parameter field exceeds 8 characters, IERR is set to -1; if there is no nonblank character before the end of the string, it is set to 1; otherwise, it is set to 0. The field is stored in A2 format in an array in COMMON.

ALTOBN (LINT, LDEC) (Fig. B-5) converts a field in the form stored by COLLECT to a binary double precision integer with scale factor using the REX service ATN. LINT is set to TRUE if the absolute value of the number is less than 32768 (i.e., if it can be contained in a single precision integer); LDEC is set to TRUE if a decimal point occurred in the number. The resulting number is placed in the same array as the input field.

INTFP (Fig. B-6) converts a number in the form stored by ALTOBN to a Modcomp double precision floating point number.

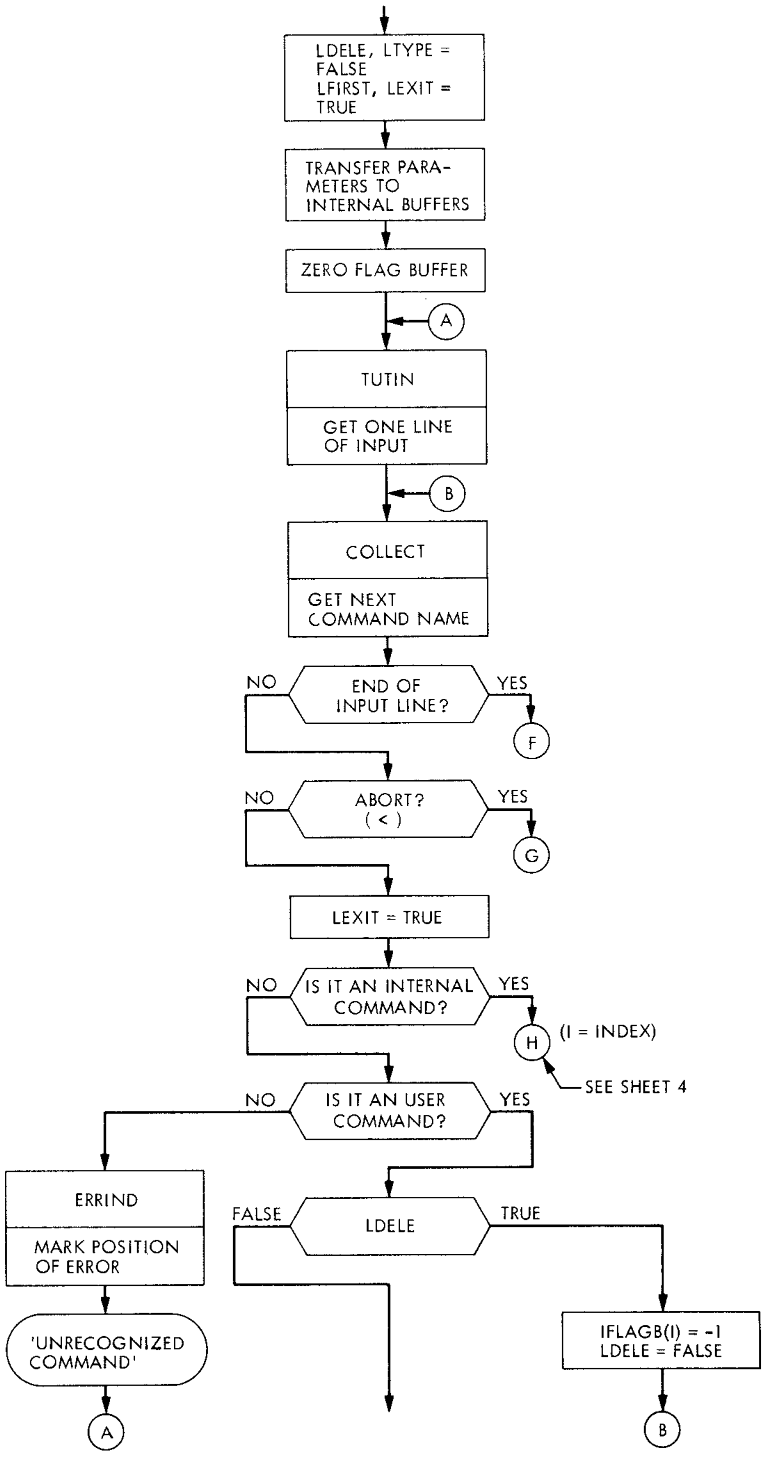
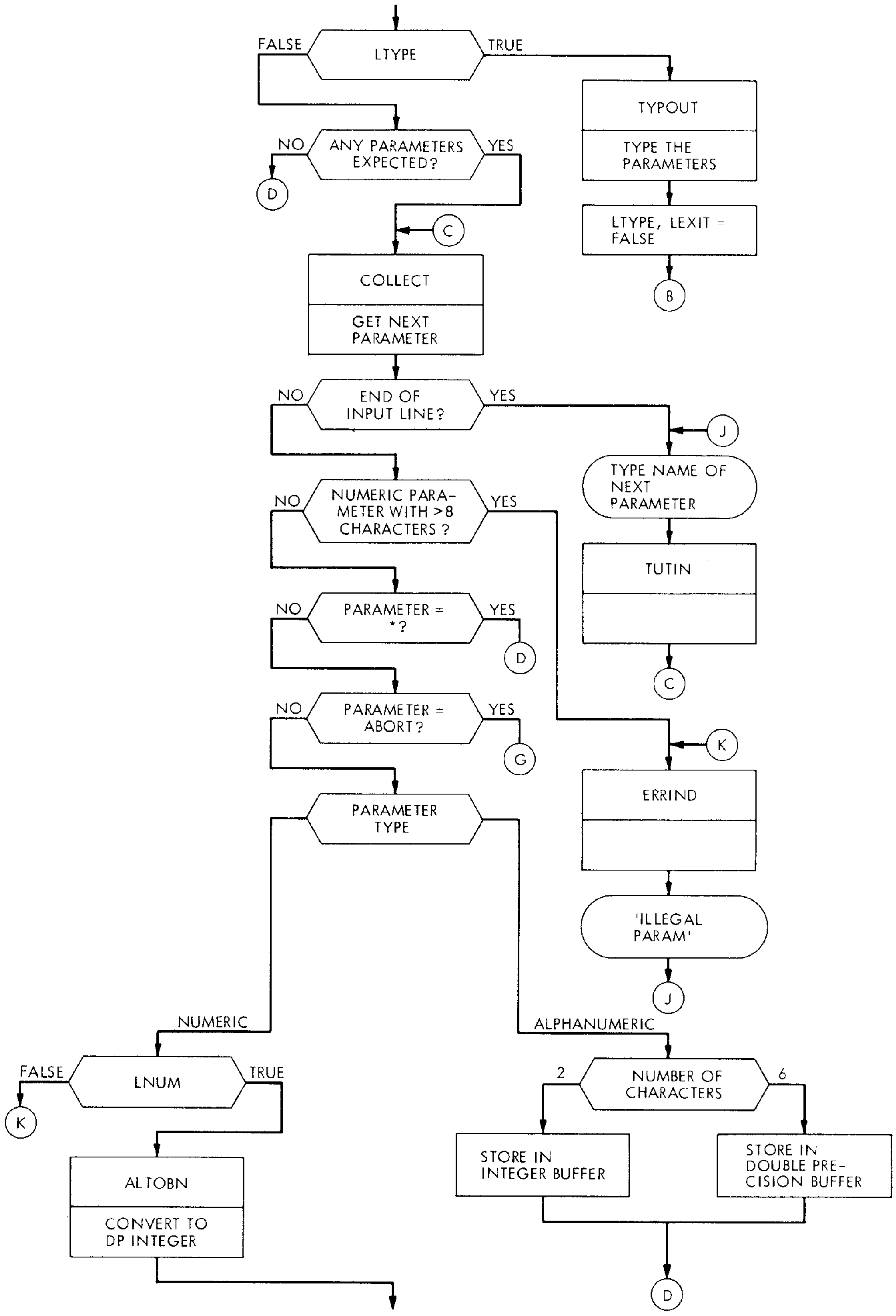

LDELE, LTYPE = FALSE
LFIRST, LEXIT = TRUE
TRANSFER PARAMETERS TO INTERNAL BUFFERS
ZERO FLAG BUFFER
A
TUTIN
GET ONE LINE OF INPUT
B
COLLECT
GET NEXT COMMAND NAME
NO
END OF INPUT LINE?
YES
F
NO
ABORT? (<)
YES
G
LEXIT = TRUE
NO
IS IT AN INTERNAL COMMAND?
YES
H
(I = INDEX)
SEE SHEET 4
NO
IS IT AN USER COMMAND?
YES
ERRIND
MARK POSITION OF ERROR
'UNRECOGNIZED COMMAND'
A
FALSE
LDELE
TRUE
IFLAGB(I) = -1
LDELE = FALSE
B

Fig. B-1. TUTOR



LTYPE
FALSE
TRUE
TYPOUT
TYPE THE PARAMETERS
LTYPE, LEXIT = FALSE
B
ANY PARAMETERS EXPECTED?
NO
YES
D
C
COLLECT
GET NEXT PARAMETER
END OF INPUT LINE?
NO
YES
J
TYPE NAME OF NEXT PARAMETER
TUTIN
C
NUMERIC PARA- METER WITH >8 CHARACTERS ?
NO
YES
PARAMETER = *?
NO
YES
D
PARAMETER = ABORT?
NO
YES
G
K
ERRIND
'ILLEGAL PARAM'
J
PARAMETER TYPE
NUMERIC
ALPHANUMERIC
LNUM
FALSE
TRUE
K
ALTOBN
CONVERT TO DP INTEGER
NUMBER OF CHARACTERS
2
6
STORE IN INTEGER BUFFER
STORE IN DOUBLE PRE- CISION BUFFER
D

Fig. B-1 (contd)

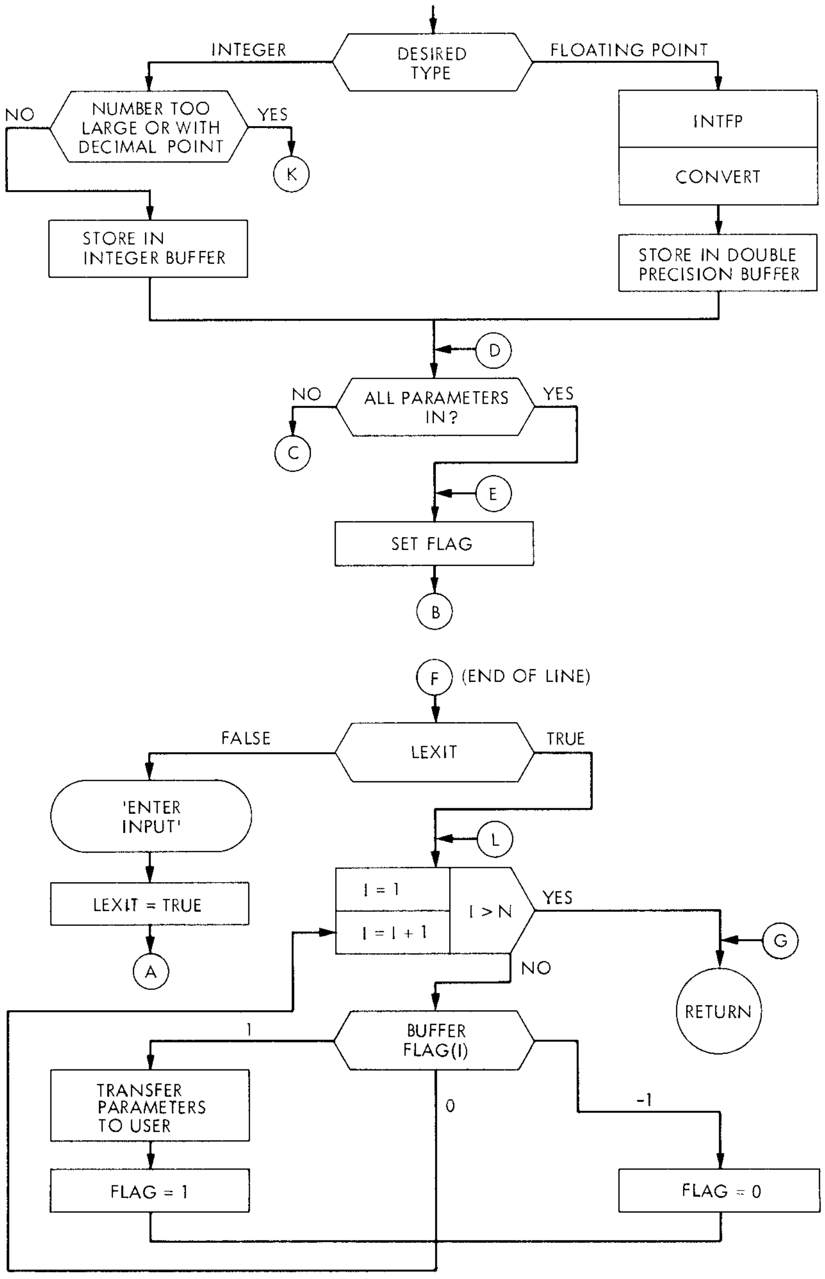

DESIRED TYPE
INTEGER
FLOATING POINT
NUMBER TOO LARGE OR WITH DECIMAL POINT
NO
YES
K
INTFP
CONVERT
STORE IN INTEGER BUFFER
STORE IN DOUBLE PRECISION BUFFER
D
ALL PARAMETERS IN?
NO
YES
C
E
SET FLAG
B
F
(END OF LINE)
LEXIT
FALSE
TRUE
'ENTER INPUT'
LEXIT = TRUE
A
L
I = 1
I = I + 1
I > N
YES
NO
G
RETURN
BUFFER FLAG(I)
1
0
-1
TRANSFER PARAMETERS TO USER
FLAG = 1
FLAG = 0

Fig. B-1 (contd)

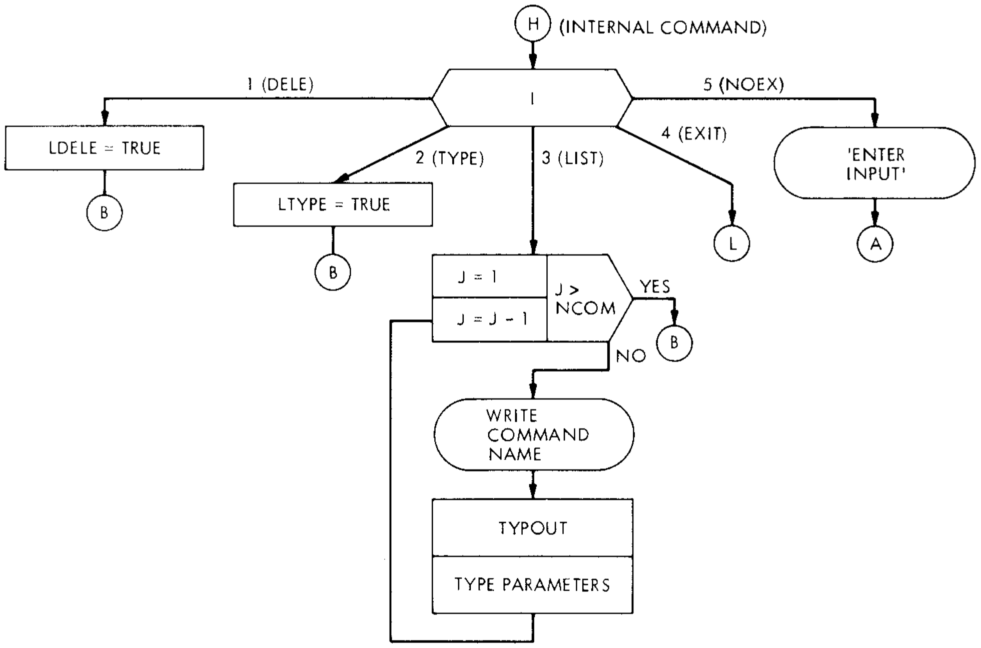


Fig. B-1 (contd)

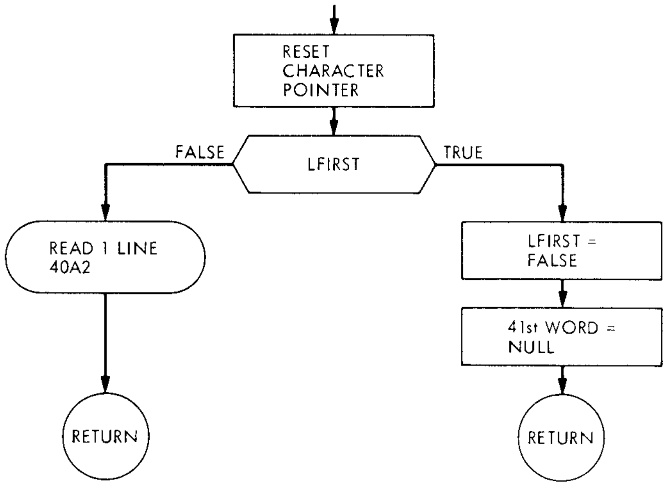


Fig. B-2. TUTIN (LFIRST)

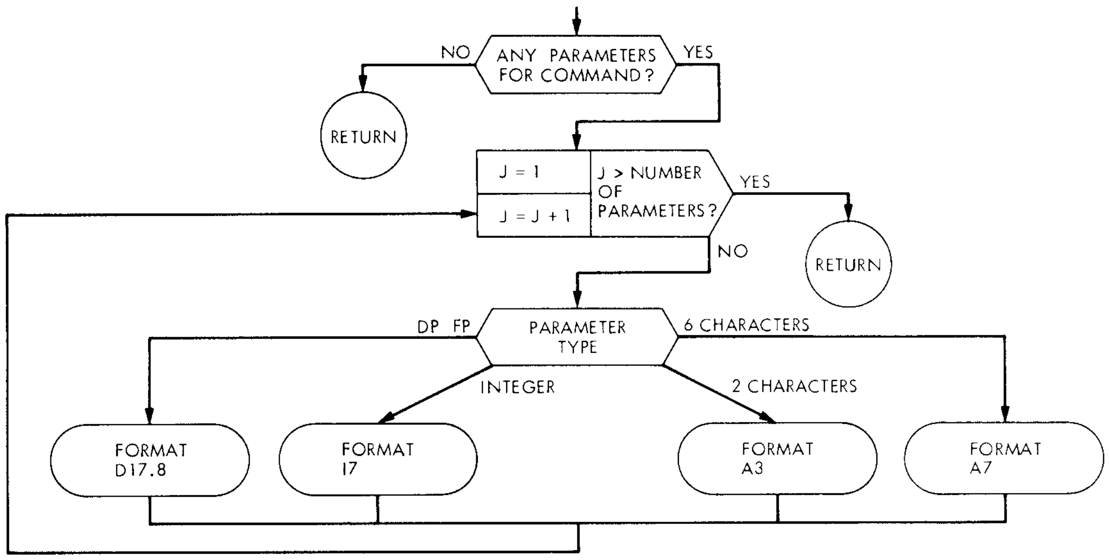


Fig. B-3. TYPOUT (I, parameter buffers)

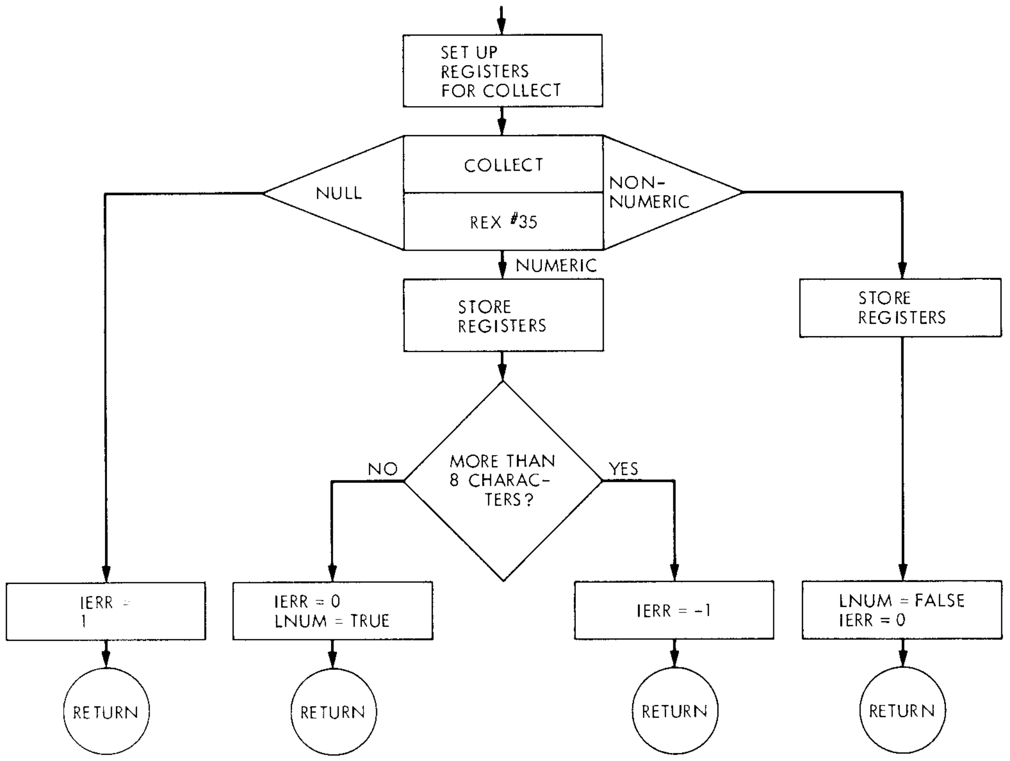


Fig. B-4. COLLECT (IERR, LNUM)

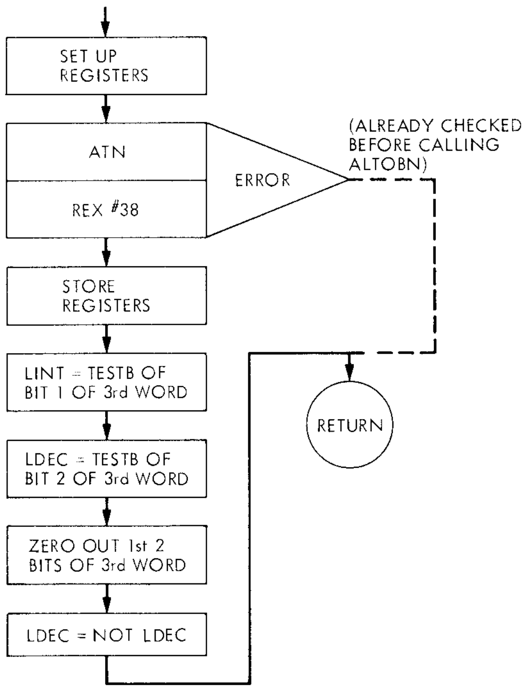

SET UP REGISTERS
ATN
REX #38
ERROR
(ALREADY CHECKED BEFORE CALLING ALTOBN)
STORE REGISTERS
LINT = TESTB OF BIT 1 OF 3rd WORD
LDEC = TESTB OF BIT 2 OF 3rd WORD
ZERO OUT 1st 2 BITS OF 3rd WORD
LDEC = NOT LDEC
RETURN

Fig. B-5. ALTOBN (LINT, LDEC)

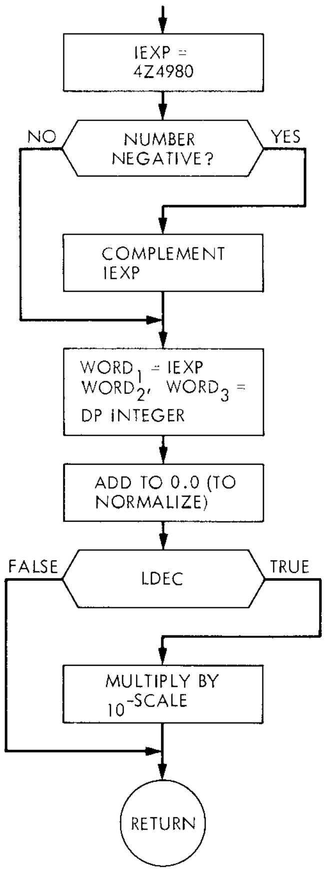


Fig. B-6. INTFP